Card Game Language Reference Manual

Jeffrey C Wong
Jcw2175@columbia.edu

1 Introduction
This language allows the programmer to create a card game with relative ease. The only thing that needs to be implemented is rules. Data structures necessary to hold card data already exist and do not require declaration.
The programmer only needs to provide value to the cards, implement conditionals and groupings.

2 Lexemes

2.1 Identifiers
Identifiers identify names of functions, class names etc. Identifies is usually a string of one or more characters consisting of upper case or lower case letters of the alphabet. Numbers can be also part of the identifier, however the identifier much start with a letter of the alphabet. Keywords are reserved identifiers and cannot be redefined.

2.2 Keywords
Shuffle
Pass
Discard
Reveal
All
Print
Total

Number
Suit
Color
Showing
Group
Misc
if
while

2.3 Numbers
Numbers in this language will only consist so a string of one or more consecutive digits. Only integers will be supported in this language.

Example 0, 1123, 42

2.4 String
String is a sequence of one or more characters. String literals are represented by surround the string with double quotes (eq. “a_string”). String can include any valid character include white spaces.

2.5 Operators

The following operators are used:

+ (add)
- (subtract)
> (greater than)
= (equal to)
< (less than)
!= (not equal to)
← (assignment)
& (and)
| (or)

2.6 Comments

Comments are initialized at the very beginning of the code line. The character to initialize a comment line is #. Any string of characters preceding # character will be considered a comment.

Ex:
this is a comment
so # is this

3 Data Types

The language contains the following data types:
Boolean: true or false
Card: C-like struct holding data types attributing to the characteristic of the card type.
Num: integers
String: A sequence of characters

4 Declarations

4.1 General declaration

Object must be declared. The syntax of declaration is as follows:

type initial-identifier();

Types can be int boolean or String.

4.2 Arrays

Arrays are created with square brackets []. An generic example:

type-specifier declarator_name[size_of_array];
The array can be accessed by
name_of_array[index]

Similarly, the index of the array can be set by assigning a value to that index.

5 Functions

5.1 Function Definitions

A function can be referenced from other source files. A body is included in the definition of a function and must return a valid type. A function may or may not include parameters.
A function definition has the following syntax:

Type function-declaration-name (parameters) {body}

Types can be int, boolean or String.

5.2 Function Declarations

Functions do not have to be declared before calling.

5.3 Function Arguments

The number of arguments must be the same as the number of parameters in order for the function to be called. Arguments in a function is separated by a commas. Arguments are passed by value.

5.4 Predefined Functions

5.4.1 Print()

Outputs to screen the parameters in the () enclosure.

5.4.2 Main()

A Specialized function that all programs begin from.

5.4.3 Shuffle()

Shuffles the decks and begins the program. No parameters are used in this function

5.4.4 Pass()

Pass passes the cards out to the grouping. Pass takes in one parameter, that is the group of where the card is going to.

5.4.5 Discard()

Removed the card in the parameter and places it in the discard. This function takes in one parameter which is the card to discard.

5.4.6 Reveal ()

Changes the value of the card to reveal. Allows for other groups to know the value of the card and in which group it belongs to.

5.4.7 Total()

Incase of addition, this function all the value of the cards in the grouping. The parameter is the grouping of the cards.

5.5 Function return

The return statement is used to return the function to where the function was initially invoked. The return statement returns a value.

6 Operators

6.1 Additive Operators

The operators + and – perform addition and subtraction. Both Operators are used only on numeric values. The result of these operators is an integer.

6.1.1 Relational Operators

The relations operators > and < perform the operation GREATER THAN and LESS THAN respectively. Operands must be of arithmetic type. The result of the expression is true or false.

6.1.2 Equality Operators

The operators == and != perform EQUALS TO and NOT EQUALS TO. The results of these operations are true and false. The == operator will produce and true result if both operands equal and false if the operands do not equal. The != operator produces the opposite result of == operator with the same operands.

6.1.3 Logical Operators

The operators & and |. The operations are AND and OR respectively. The results of the operators is true or false. The compiler is left associative. If the left expression is true for operator |, then the result is true. Otherwise the compiler will check the right expression. If both expression aren't true, then the result if false. The & operator checks if either left or right expression is false and returns true if both are not false.

6.1.4 Assignment Operators

The operator ← is the assignment operator. The value to the right of the operator is assigned to the left of the operator.
Ex:

Expr1 ← Expr2